Writing Style Mimic Engine - Full Code Explanation

This project is a web application that can:

1. **Generate text** in different writing styles (formal, casual, technical, etc.)

2. **Compare writing styles** between two texts to determine similarity and potential authorship

Let me break down each component in detail:

1. Frontend (HTML/CSS/JS)

index.html

The main interface with 3 sections:

- **Generate Text**: Input prompt + style selection

- **Compare Texts**: Two text areas for comparison

- **About**: Project description

Key features:

- Responsive design works on all devices

- Interactive elements (buttons, dropdowns)

- Loading indicators during processing

- Copy/download functionality for generated text

styles.css

Provides:

- Modern, clean UI with cards and shadows

- Color scheme using CSS variables

- Responsive layouts (flexbox/grid)

- Animations for better UX
- Toggle switches and form styling

script.js
Handles:
- Form submissions (generation/comparison)
- API calls to backend
- Dynamic UI updates
- Error handling
- Simulated responses (when backend not available)

2. Backend (Python/Flask)

app.py
Flask server with 3 main endpoints:
1. `/api/generate` - Takes prompt + style → returns generated text
2. `/api/compare` - Takes two texts → returns similarity analysis
3. `/api/save_model` - (Placeholder) for saving custom models

Features:
- CORS enabled for frontend-backend communication
- Error handling for malformed requests
- Model initialization on startup

model.py
Core AI functionality with 2 classes:

TextGenerator:

- Style templates for each writing style
- Fill-in-the-blank generation system
- Can extend with real ML models

StyleAnalyzer:
- Uses TF-IDF vectorization
- Cosine similarity for style comparison
- Heuristics for authorship prediction
- Can be replaced with more advanced NLP

utils.py
Supporting functions for:
- Text preprocessing (cleaning, normalization)
- Feature extraction (word counts, sentence stats)
- N-gram generation
- Similarity calculations

3. Project Infrastructure

requirements.txt
Lists all Python dependencies:
- Flask (web framework)
- scikit-learn (ML utilities)
- numpy (numerical operations)
- python-dotenv (environment variables)

.gitignore
Prevents committing:

- Virtual environment files
- Python cache files
- Model binaries
- IDE configurations

Key Algorithms

1. **Text Generation**:
   - Template-based approach (for demo)
   - Could integrate GPT-2/3 for production

2. **Style Analysis**:
   ```python
   # Vectorize texts
   vectors = vectorizer.fit_transform([text1, text2])

   # Calculate similarity
   similarity = cosine_similarity(vectors[0], vectors[1])[0][0]
   ```

3. **Authorship Prediction**:
   - Threshold-based (65%+ similarity = same author)
   - Confidence score based on similarity distance

Workflow

1. User submits prompt + style selection
2. Frontend sends request to `/api/generate`

3. Backend processes with TextGenerator

4. Response returns generated text

5. UI updates with formatted output

For comparison:

1. User inputs two texts

2. Frontend sends to `/api/compare`

3. Backend analyzes with StyleAnalyzer

4. Returns similarity metrics

5. Frontend visualizes results

Scalability Options

To enhance this project:

1. **Replace generators** with transformer models (GPT, BERT)

2. **Add authentication** for user-specific style profiles

3. **Implement model training** endpoint

4. **Add database** for saving comparisons

5. **Deploy with Docker** for production

Would you like me to dive deeper into any specific component? For example:

- How the cosine similarity calculation works

- Ways to improve the template-based generation

- How to connect a real ML model

- Deployment strategies